Truck Platooning: week 4

Changyao Zhou, Jiaxin Pan

January 17, 2022

1 Introduction of the project

The goal of this project is to chase a chosen target vehicle by predicting the steering angle and throttle input of the ego-vehicle, given the velocity of both vehicles and images taken from cameras.

The control values (steering angle and throttle) computed by Model Predictive Control (MPC) are used as ground-truth. MPC is an advanced method of process control that is used to control a process while satisfying a set of constraints[WIK]. It takes a prediction horizon instead of a single time step into account, and aims to get an optimal control result minimizing the objective function within the prediction horizon. For learning the MPC controller, we take [MPC] as reference.

The project is designed to be realized in the following steps:

1). Control the ego-vehicle with Model Predictive Control (MPC) in the Carla simulator

2). Given x, y-location, yaw angle and velocity of both vehicles, train a fully connected neural network to control ego-vehicle by using the MPC output as ground-truth.

2). Given object velocity, ego-velocity and RGB image, train a neural network to control the ego-vehicle by using the MPC output as ground-truth

3). Further improvement by changing the input of the neural network

2 Overview of this week

Last week we have implemented the online MPC controller in CARLA simulator, which means we could use the control values (throttle and steering angle) computed by MPC to control the ego-vehicle in real time to follow the target vehicle. Then we started to collect data for training NN, including data points exactly on the planned trajectory and data points in the neighborhood.

This week, we continued to collect more data. We first tried to collect data in 'offline' mode without CARLA simulator. But later we found it's difficult to restore the motion model of the car in the simulator. So we changed to collect data in 'online' mode within CARLA simulator. After that we started to build and train the model. At last, we put the trained model back in the simulator to do inference. The inference result seems to be stable and could successfully control the ego-vehicle to follow the target vehicle.

All the videos are stored under https://syncandshare.lrz.de/getlink/fiBHjTqaEdCQFCsVcF2iuuC7/week4.

3 Data Collection

The entire dataset contains the data points exactly on the planned trajectory and data points in the neighborhood. And for each single sample, the input includes the x- and y-position, yaw angle, and velocity of both ego-vehicle and target vehicle. To get enough data for model robustness, a large mount of data points around the planned trajectory should be collected, which is really time-consuming.

3.1 Offline Data Collection

To save time, we first tried to collect data without CARLA simulator. To be more specific, we collected data points on the trajectory for both target vehicle and ego-vehicle in CARLA simulator. Then we could get out of CARLA. During the collection, for each ego-vehicle state, we sampled a set of points around its position and also changed its velocity and yaw angle as disturbance. Then the disturbed ego-vehicle should follow the step of the target vehicle in corresponding position with help of MPC controller. **The state of ego-vehicle is updated by plant model.** During the process of getting back to follow the target vehicle, the state of both vehicles and the output control values of MPC at each time step are collected as input and labels respectively. The general effect of collecting data is shown in the Figure 1.

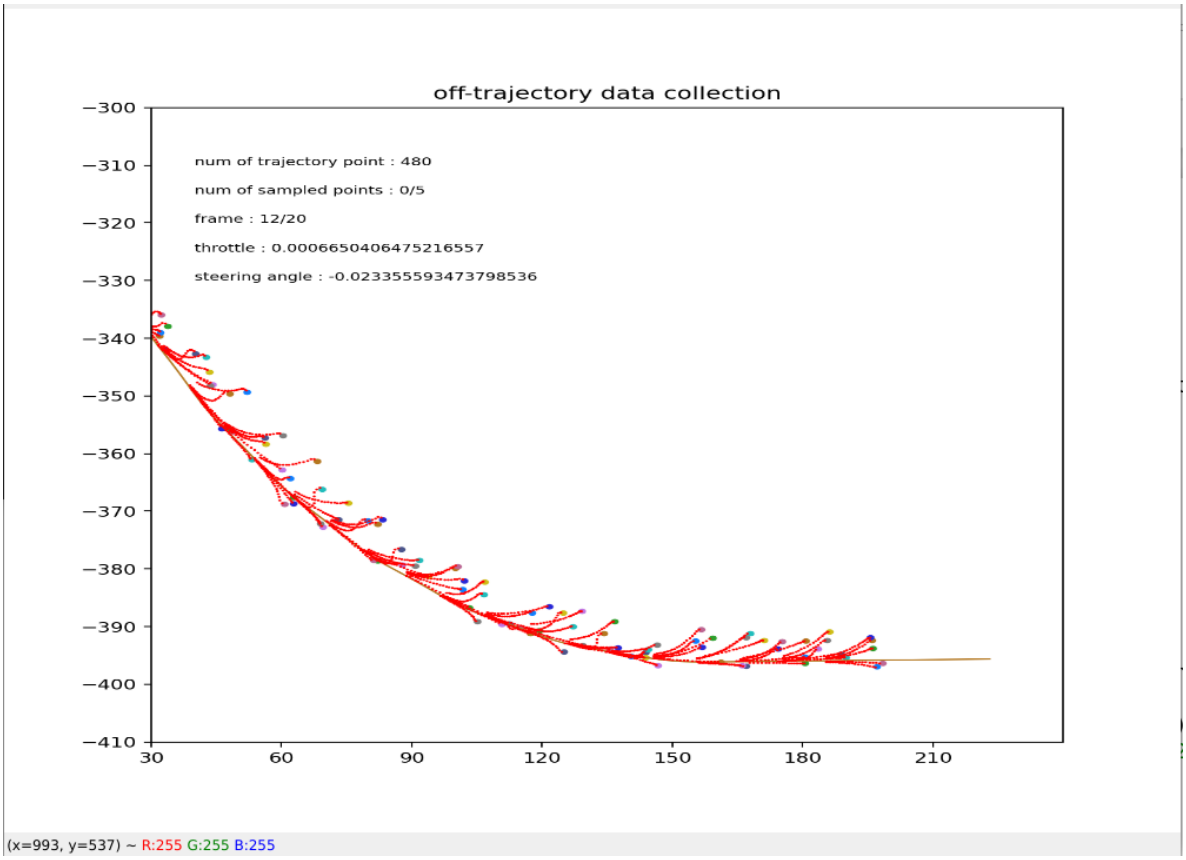

Figure 1: Few Data points collected around the planned trajectory offline

In this way, we could collect many data points around the original trajectory. However, when we tried to put the collected data back to the CARLA simulator, we found that the collected control values can't let the ego-vehicle keep up with the target vehicle. The problem is that our mathematical plant model can't really accurately describe the motion model of the car in CARLA simulator. Also, it's hard to adjust the plant model to fit various situation during following process. That's why we have to turn to collect data online within the CARLA simulator.

3.2 Online Data Collection

For online data collection, to get a stable trajectory as before, we used the control values collected before during the autopilot mode to control the target vehicle. And our basic idea is generally similar to offline collection mentioned above. One of the differences is that **the states of both vehicles are updated by the simulator itself.** And we could use the function *get_transform()*, *get_velocity()*, *get_location()*, to read the real-time state of both vehicles and feed them into MPC controller to get more precious control values.

As the collection process mentioned above, we chose to start at frame 40 and do the sampling process every 10 frames (at frame 40, 50, ...). And for each frame on the trajectory, we randomly sampled 20 states with offsets and disturbances (change of position, yaw angle, velocity of ego-vehicle) for the ego-vehicle. Then the ego-vehicle would start at these different states and try to follow the target vehicle. For each state, the control values from MPC controller as well as states of both vehicles in following 20 frames are collected as data around trajectory. In this way, the ego-vehicle could generate more trajectories around, which could cover almost all areas in the neighborhood of the planned trajectory. The collected trajectories from frame 650 to frame 800 are shown in the Figure 2 below.

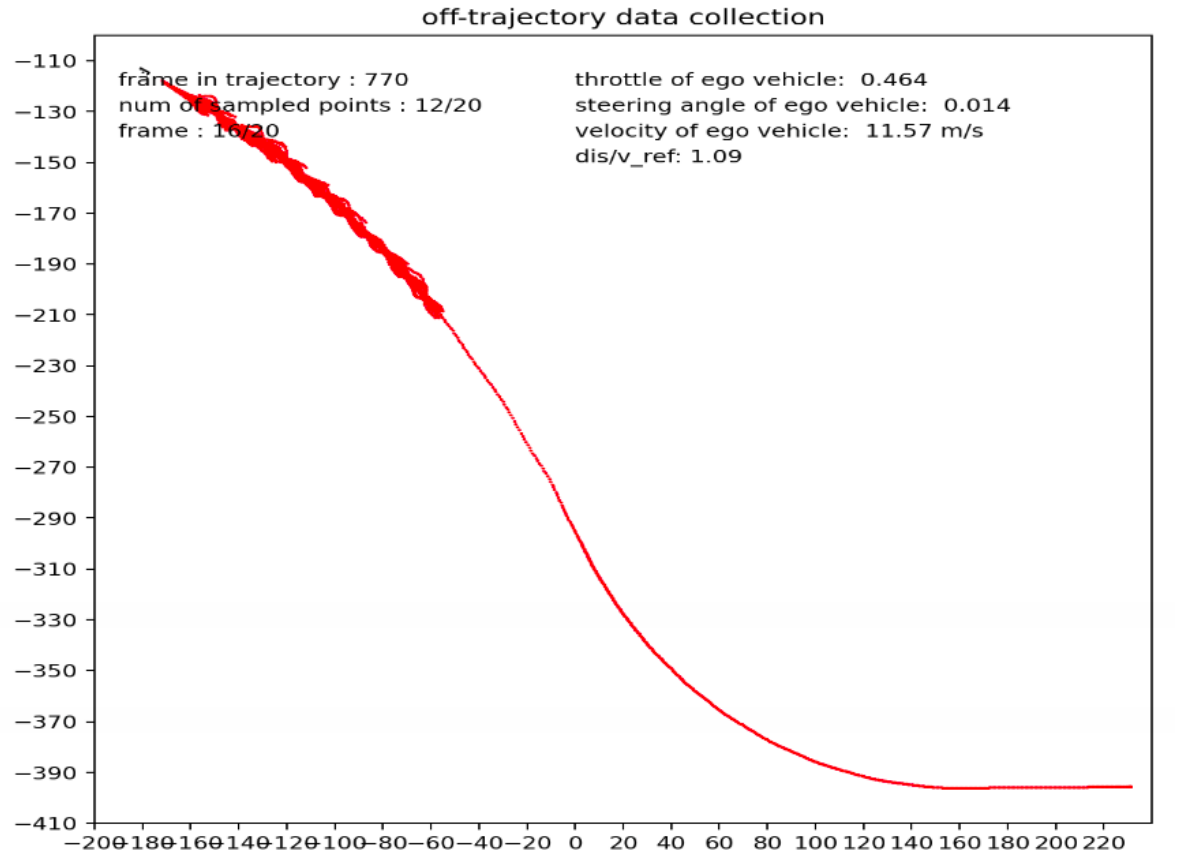

Figure 2: Data points collected around the planned trajectory online

In this way, we have collected around 30000 samples in the neighborhood of the planned trajectory, which could improve the robustness of model and help the ego-vehicle come back on track in time when the car drifts. There's a video for a shortcut of the collection of off-trajectory samples. https://syncandshare.lrz.de/getlink/fiPg9L8YHbPAJ35Hm18Jvoz8/off_collection.mp4

4 Training of FCN Model

After collecting dataset with both on- and off-trajectory data, we could start to create the neural network and train it. Here we used a 6-layer NN with batch normalization and the activation function is Relu. The network contains 4 million parameters in total.

We first used only two samples to train it. And it turns out that the NN overfits, which indicates the NN architecture is reasonable. Then we started to train it for a total of 6000 epoches with two different learning rates in sequence, 1e-4 and 1e-6. The loss curve is shown in the Figure 3. Finally, the training loss reaches 1e-2.

After training, we could use the trained model to do inference in CARLA simulator and see if it works well. With the real-time state of both target vehicle and ego-vehicle read from CARLA, the

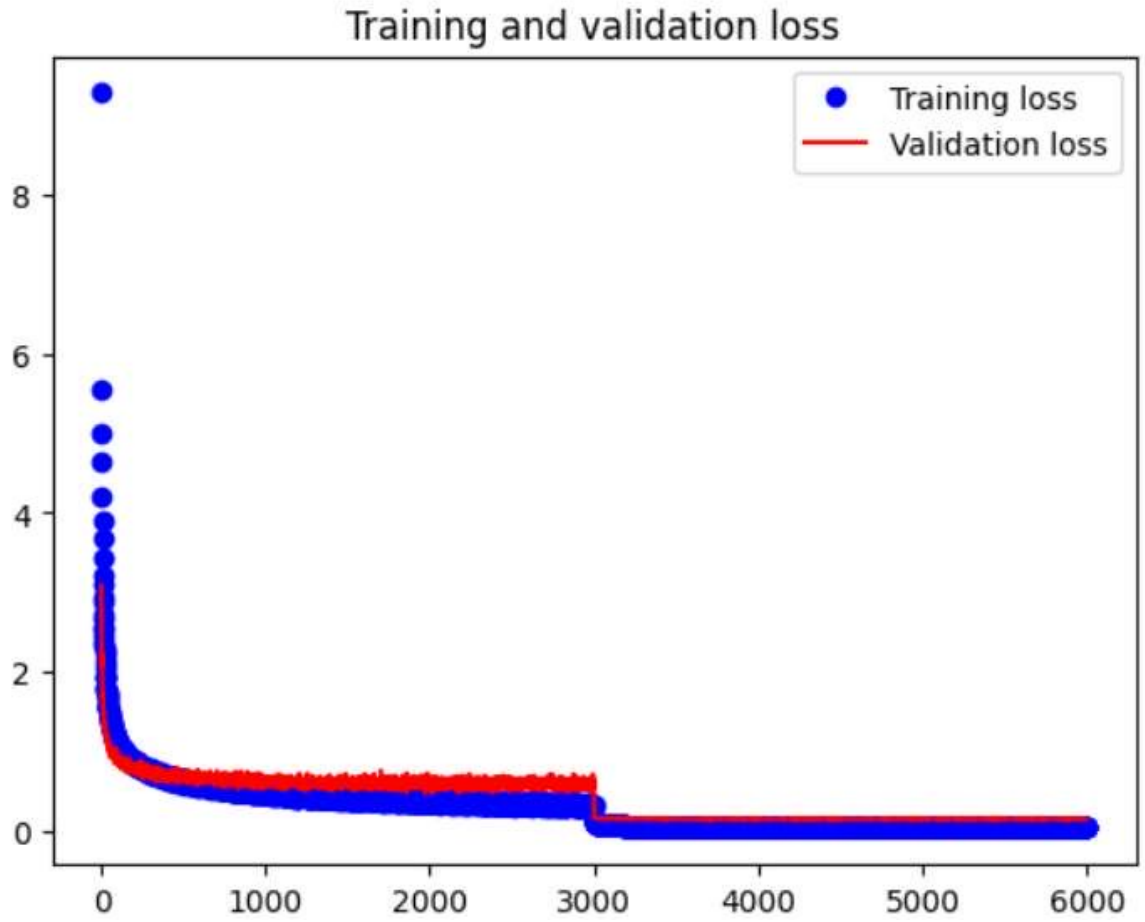

Figure 3: The loss curve during 6000-epoch training

output of NN can be computed, the two output values throttle and steering angle are used to control the ego-vehicle to follow the target. And the result seems good, which means the trained model could make accurate predictions for control values.

The video for car following with model inference can be found here: https://syncandshare.lrz.de/getlink/fiMQdfCMmK3Kg9mzMHCeVzUf/network_test.mp4

From the video we can also see, that the ratio between the car distance and the velocity of target vehicle keeps relatively constant. That means the car following process is stable and perform well.

To test the stability of the network, we added some random impulses on the ego vehicle every 40 frames with the equation (1) [ADD] to test if the network can figure out the control values that can get the ego vehicle back to the trajectory. The result can be seen in Figure 4. With the random impulses, the network can still control the ego vehicle following the target vehicle, which means the output is quite stable. The video of the test can be checked here:https://syncandshare.lrz.de/getlink/fiVJ21Lxi6SpwUNojxqAyGnf/random_impulses.mp4.

$$vehicle.add\_impulse(carla.Vector3D(impulse, impulse, 0)) \tag{1}$$

5 Conclusion and future work

In this week, we have collected dataset for both on- and off-trajectory samples and trained the FCN model to predict the control values. The current inputs of model are states including x-,y-position, yaw angle, and velocity of both vehicles. Our next step will be only keeping the velocity and replacing the rest three parameters with images, which means only the velocities and images would be feed into

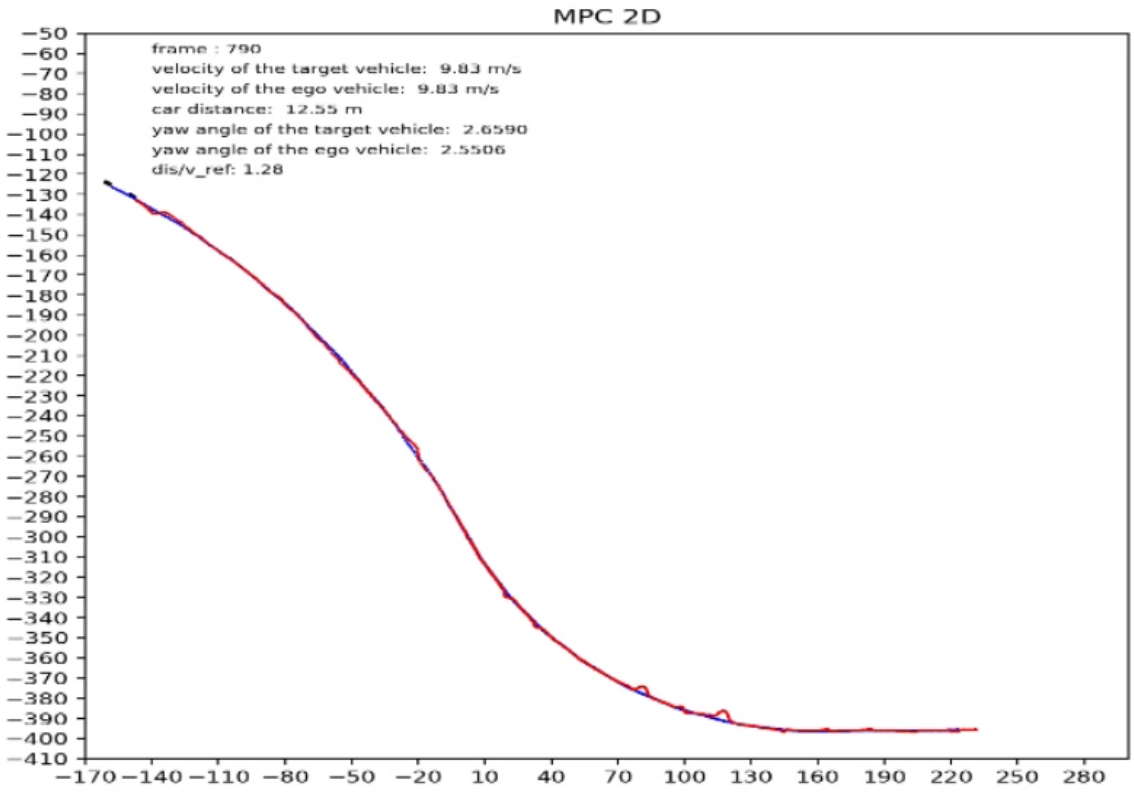

Figure 4: The test with random impulses.

the model as input. And if it works well, the input of the model will be further simpler to implement a more automatic car following process.

References

[ADD] Carla Add Impulse. https://github.com/carla-simulator/carla/blob/8854804f4d7748e14d937ec763a2912823a7e5f5/Docs/python_api.md#method.

[MPC] Model Predictive Control. https://github.com/WuStangDan/mpc-course-assignments.

[WIK] Model Predictive Control. https://en.wikipedia.org/wiki/Model_predictive_control.